Individual Report

Jia Chen

1. Introduction

Facial expression is a primary way to convey emotions and intentions of human beings. Detecting facial expression promotes communication between individuals. With the development of technology, facial expression recognition has attracted special research attention. A large volume of studies has been focused on "automatic facial expression analysis because of its practical importance in sociable robotics, medical treatment, driver fatigue surveillance, and many other human-computer interaction systems" (Li & Deng, 2018). In the field of machine learning, facial expression recognition has been well developed due to the improvement of database and deep leaning techniques.

This project aims to train a deep facial expression recognition model that can classify given facial image into one of seven classes. We select the Real-world Affective Face Database (RAF-DB) as data. Different models (CNN and MLP) are implemented using TensorFlow (Keras API). In the process of optimize model to get better accuracy, we also explore how hyperparameters influence the performance of network. Finally, the performances of these models are compared. I am responsible for implementing MLP using Keras.

2. Neural network and Framework

2.1 Multilayer perceptron

The first deep neural network used in this project s multilayer perceptron (MLP). MLP is a sequential artificial neural network. As shown in figure 1, in general, MLP is composed of three parts: input, hidden layer, and output layer. There can be more than one layer inside of hidden layer. There must be transfer function for each layer. After dot product and transfer function of first layer, the output of first layer becomes the input of second layers. In output layer, the output of hidden layer will be classified predicted or classified. MLP utilizes a supervised learning technique called backpropagation for training.

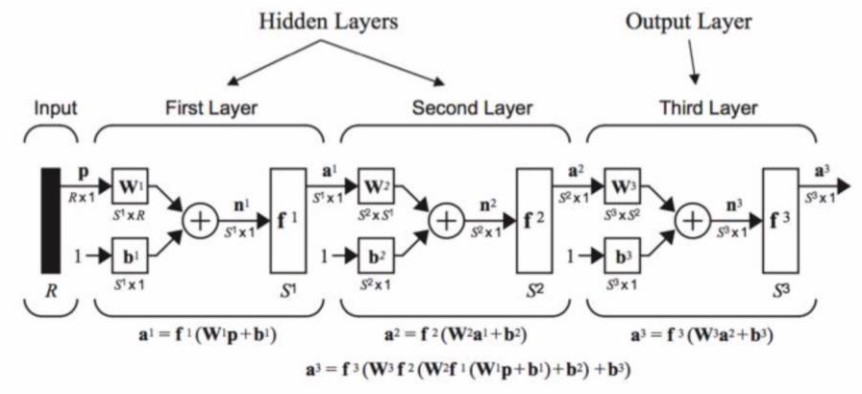

Figure 1. Typical architecture of MLP

2.2 TensorFlow (Keras API)

All the models in this project are implemented by Keras. “It is an open source neural network library written in Python”. With the development of deep neural network, adjusting parameters has been becoming a more and more difficult task when optimizing network. Keras makes it easier by being user friendly, modular, and extensible. “Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizes, and a host of tools to make working with image and text data easier”. Keras supports many neural networks such as MLP and CNN.

3. Description of individual work

All the models I defined are in MLP_model.py file then import them in MLP_main.py file.

In this section, I will discuss and show how parameters, including optimizer, number of epochs, learning rate, dropout rate, batch size, and number of layers influence the performance of MLP.

We create MLP models by assembling layers. In Keras, Dense is used to generate layers and provide output for next layer. As shown in figure 7, there are two layers in total. The

number of neurons in hidden layer can be set as needed. However, the number of neurons in output layer must equal the number of class. In our project, it should be 7 because there are 7 kinds of emotions. The first layer of MLP requires the input dimension (shape), not including batch size. Therefore, we put 30000 because data are 100*100 colored images. Also we selected mini batch of 32 in this study for the reason that both the dimension (30000) of input and number of training image (12271) are not small. For transfer function, ReLU is used in hidden layer due to its sparsity and makes neural network run faster. Softmax is chosen for output layers because we aim to solve a classification problem.

```
def m_1():
    model = tf.keras.Sequential([
        layers.Dense(hidden1_num, activation='relu', input_dim=30000),
        layers.Dense(output_num, activation='softmax'),
    ])
```

Figure 7. Dense

Before training the model, a few more settings can be implemented by compile. Set loss to categorical crossentropy. Then Adam optimizer is selected in first model. Different optimizer will be compared below.

```
    model.compile(optimizer=tf.train.AdamOptimizer(0.001),
                  loss='categorical_crossentropy',
                  metrics=[tf.keras.metrics.categorical_accuracy])

    return model
```

Figure 8. Compile

Based on above statement, in the same condition (batch_size = 32, num_epoch = 5, num_layer = 2, hidden_num = 50, lr = 0.001), the accuracy of different optimizers is listed in table 1. It is clear that SGD works better in this network. So, we select SGD.

Optimizer	Adam	Adadelta	RMSProp	SGD
Accuracy	22.1%	38.3%	38.5%	55.2%

Table 1. Optimizer

Next, just keep all the parameters but increase the number of epochs to 50, and add

dropout with rate of 0.2 to prevent overfitting. The results are shown in table 2. It suggests that with the increase of epochs, the accuracy increases obviously.

lr	Dropout	10	20	30	40	50
0.001	0.2	0.5968	0.6196	0.6231	0.6349	0.6643

Table 2. Accuracy for different epochs_lr_0.001

In order to make the network converge faster, we consider changing learning rate larger to 0.01. The results are shown in table 3. Despite with expectation, larger learning rate cannot promote the performance of our model. It proves that step size of 0.01 is too big for this network that leads to skip the minimum points. So we keep the learning rate as 0.001.

lr	Dropout	10	20	30	40	50
0.01	0.2	0.5209	0.528			0.6379

Table 3. Accuracy for different epochs_lr_0.01

Adjusting mini batch size will influence model in many aspects. If we increase the size of mini batch, the iteration of one epoch will reduce, thus speed up the model for each epoch. To get the same accuracy with smaller batch size, larger size of mini batch needs more epochs. What's more, larger mini batch size requires larger memory capacity. We tried different batch size. According to results, the size of 32 is the best choice.

Batch size	32	64	128
Accuracy	0.6780	0.6670	0.6385
Time	123	132	137

Table 4. Accuracy for different batch size, lr_0.001

All the above discussion is based on a 2-layer perceptron. To get better performance, adding layer is another method. Keep the number of neuron in first layer as 50, adding layers to create different models as table 5. With the increase of layers, the accuracy declined. Therefore, 2-layer architecture is ideal for this data.

Hidden layer					Output layer	Accuracy
L_1: 50					7	0.5968
L_1: 50	L_2: 10				7	0.3862
L_1: 50	L_2: 10	L_3: 10			7	0.3862
L_1: 10	L_2: 10	L_3: 10	L_4: 10	L_5: 10	7	0.3862

Table 5. Accuracy for different number of layer, epoch=10

To summary, we adjusted certain parameters by controlling other parameters and got different types of MLP. In the result section, we will summarize and point out the best MLP for this dataset.

4. Result

According to discussion in section 4, it is suggested that model "m_3" (can be found in python file called MLP_model.py) with batch size as 32 performances best for this dataset. As shown in following table 6, the accuracy reached 67.8%. The same model with batch size of 32 got a slightly smaller accuracy of 66.7%. Figure 9 and figure 10 shows the change of validation loss and validation accuracy. The blue line represents batch size 32 and red line stands for batch size 64.

Model	Batch_size	Num_epoch	layer	Num_hidden	Num_output	Dropout	lr	Optimizer	Accuracy
m_3	64	60	2	50	7	0.2	0.001	SGD	66.7%
	32	60	2	50	7	0.2	0.001	SGD	67.8%

Table 6. Ideal models

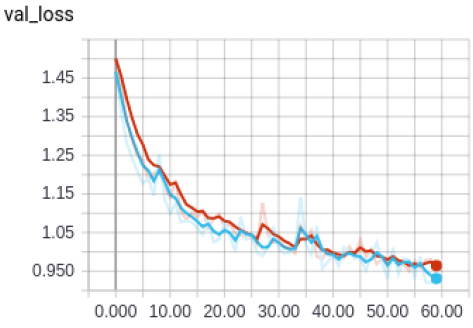

Figure 9

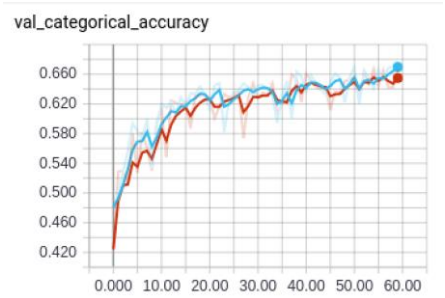

Figure 10

5. Summary

The result shows that multilayer perceptron is not a good model for expression classification on RAF-DB, as the accuracy is only 67.8, especially compared to the accuracy of convolutional network: 74%. In the future we will try more different combinations of parameters.

6. Code calculation

MLP_helper_C.py - 68%

MLP_main.py - 71%

MLP_model.py - 0%

MLP_helper_C.py and MLP_main.py is adjusted from MLP_helper.py and MLP_main.py which are by Xiaochi Li.

7. Reference

S.Li and W.Deng, “Deep Facial Expression Recognition: A survey.” ArXiv, 2018.